```
package main

import "testing"

func FiboRecursive(n uint64) uint64 {
      if n < 2 {
            return 1
      } else {
            return FiboRecursive(n-1) + FiboRecursive(n-2)
      }
}

func FiboCyclic(n uint64) uint64 {
      a, b := uint64(1), uint64(0)  // a = f[0], b = f[-1]
    for n > 0 {
        a, b = a+b, a
        n--
    }
    return a
}

type memo []uint64

func F(n uint64, m memo) uint64 {
    if m[n] == 0 {
        m[n] = F(n-1, m) + F(n-2, m)
    }
    return m[n]
}

func FiboMemo(n uint64) uint64 {
    m := make([]uint64, n+1, n+1)
    m[0], m[1] = 1, 1
    return F(n, m)
}

// Названия тестируемых функций должны начинаться на Benchmark,
// за которым идёт название, начинающееся с большой буквы

func BenchmarkFiboRecursive16(b *testing.B) {
    for i := 0; i < b.N; i++ {
        FiboRecursive(16)
    }
}

func BenchmarkFiboRecursive24(b *testing.B) {
    for i := 0; i < b.N; i++ {
        FiboRecursive(24)
    }
}

func BenchmarkFiboRecursive32(b *testing.B) {
    for i := 0; i < b.N; i++ {
        FiboRecursive(32)
    }
}

func BenchmarkFiboRecursive40(b *testing.B) {
    for i := 0; i < b.N; i++ {
        FiboRecursive(40)
    }
}

func BenchmarkFiboCyclic16(b *testing.B) {
    for i := 0; i < b.N; i++ {
        FiboCyclic(16)
    }
}
```

```
69
70  func BenchmarkFiboCyclic24(b *testing.B) {
71      for i := 0; i < b.N; i++ {
72          FiboCyclic(24)
73      }
74  }
75
76  func BenchmarkFiboCyclic32(b *testing.B) {
77      for i := 0; i < b.N; i++ {
78          FiboCyclic(32)
79      }
80  }
81
82  func BenchmarkFiboCyclic40(b *testing.B) {
83      for i := 0; i < b.N; i++ {
84          FiboCyclic(40)
85      }
86  }
87
88  func BenchmarkFiboCyclic60(b *testing.B) {
89      for i := 0; i < b.N; i++ {
90          FiboCyclic(60)
91      }
92  }
93
94  func BenchmarkFiboCyclic90(b *testing.B) {
95      for i := 0; i < b.N; i++ {
96          FiboCyclic(90)
97      }
98  }
99
100 func BenchmarkFiboMemo16(b *testing.B) {
101     for i := 0; i < b.N; i++ {
102         FiboMemo(16)
103     }
104 }
105
106 func BenchmarkFiboMemo24(b *testing.B) {
107     for i := 0; i < b.N; i++ {
108         FiboMemo(24)
109     }
110 }
111
112 func BenchmarkFiboMemo32(b *testing.B) {
113     for i := 0; i < b.N; i++ {
114         FiboMemo(32)
115     }
116 }
117
118 func BenchmarkFiboMemo40(b *testing.B) {
119     for i := 0; i < b.N; i++ {
120         FiboMemo(40)
121     }
122 }
123
124 func BenchmarkFiboMemo60(b *testing.B) {
125     for i := 0; i < b.N; i++ {
126         FiboMemo(60)
127     }
128 }
129
130 func BenchmarkFiboMemo90(b *testing.B) {
131     for i := 0; i < b.N; i++ {
132         FiboMemo(90)
133     }
134 }
```

benchmark_Fibonacci_test.go

```go
package main

// Запуск из командной строки:
//         go test -bench . benchmark_Invest_test.go
// Имя файла обязательно должно заканиваться на _test

import (
    "fmt"
    "os"
    "testing"
)

var (
    inc       [][]int
    n, maxsum int
)

func init() {
    f, _ := os.Open("income.dat")
    fmt.Fscan(f, &n, &maxsum)
    // n - количество вариантов инвестиций (производств)
    // sum - максимально возможная суммарная сумма инвестиций
    // inc[i][s] - доход, который приносит инвестиция
    //             s денег в производство #i
    inc = append(inc, make([]int, maxsum+1, maxsum+1))
    for i := 1; i <= n; i++ {
        inc = append(inc, make([]int, maxsum+1, maxsum+1))
        inc[i][0] = 0
        for s := 1; s <= maxsum; s++ {
            fmt.Fscan(f, &inc[i][s])
        }
    }
}




func MaxIncomeCyclic(n int, sum int) int {
    // инвестируем sum денег в проекты 1..n
    var max, w int
    var res [][]int
    res = append(res, make([]int, sum+1, sum+1))
    for k := 1; k <= n; k++ {
        res = append(res, make([]int, sum+1, sum+1))
        max = inc[k][sum]
        for s := 1; s <= sum; s++ {
            for reminder := 1; reminder <= s; reminder++ {
                w = inc[n][s-reminder] + res[k-1][reminder]
                if w > max {
                    max = w
                }
            }
            res[k][s] = max
        }
    }
    return res[n][sum]
}

```

```
func MaxIncomeRecursive(n int, sum int) (result int) {
    // инвестируем sum денег в проекты 1..n
    var w int
    if n > 0 {
        result = inc[n][sum]
        for reminder := 1; reminder <= sum; reminder++ {
            w = inc[n][sum-reminder] + MaxIncomeRecursive(n-1, reminder)
            if w > result {
                result = w
            }
        }
        return result
    } else {
        return 0
    }
}

type memo [][]int

func MaxIncome(n int, sum int, m memo) int {
    // Рекурсивная функция с мемоизацией
    // инвестируем sum денег в проекты 1..n
    var w, result int
    if n > 0 && m[n][sum] == 0 {
        result = inc[n][sum]
        for reminder := 1; reminder <= sum; reminder++ {
            w = inc[n][sum-reminder] + MaxIncome(n-1, reminder, m)
            if w > result {
                result = w
            }
        }
        m[n][sum] = result
    }
    return m[n][sum]
}

func MaxIncomeMemo(n int, sum int) (result int) {
    // Слайс res используется для мемоизации
    var res [][]int
    // Подготавливаем слайс res, он заполняется нулями.
    for i := 0; i <= n; i++ {
        res = append(res, make([]int, sum+1, sum+1))
    }
    // Рекурсивно (с мемоизацией) вычисляем максимальный доход,
    // получаемый при инвестировании sum денег в проекты 1..n.
    return MaxIncome(n, sum, res)
}

```

```go
// Названия тестируемых функций должны начинаться на Benchmark,
// за которым идёт название, начинающееся с большой буквы

func BenchmarkInvestRecursive_5_10(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeRecursive(5, 10)
    }
}

func BenchmarkInvestRecursive_4_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeRecursive(4, 20)
    }
}

func BenchmarkInvestRecursive_5_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeRecursive(5, 20)
    }
}

func BenchmarkInvestCyclic_5_10(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeCyclic(5, 10)
    }
}

func BenchmarkInvestCyclic_4_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeCyclic(4, 20)
    }
}

func BenchmarkInvestCyclic_5_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeCyclic(5, 20)
    }
}

func BenchmarkInvestMemo_5_10(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeMemo(5, 10)
    }
}

func BenchmarkInvestMemo_4_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeMemo(4, 20)
    }
}

func BenchmarkInvestMemo_5_20(b *testing.B) {
    for i := 0; i < b.N; i++ {
        MaxIncomeMemo(5, 20)
    }
}
```

benchmark_Invest_test.go

```
package main
import "testing"

const mod = 1000000007

func Mult(a, b uint32) uint32 {
	return uint32(uint64(a) * uint64(b) % mod)
}

func PowerCyclic0(a uint32, n uint32) uint32 {
	var res uint32 = 1
	for ; n > 0; n-- {
		res = Mult(res, a)
	}
	return res
}

func PowerRecursive0(a uint32, n uint32) uint32 {
	if n == 0 {
		return 1
	}
	return Mult(a, PowerRecursive0(a, n-1))
}

func PowerRecursive1(a uint32, n uint32) uint32 {
	var b uint32
	if n == 0 {
		return 1
	} else {
		b = PowerRecursive1(a, n/2) // b = a^(n/2)
		b = Mult(b, b)
		if n%2 == 0 {
			return b
		} else {
			return Mult(b, a)
		}
	}
}

func PowerRecursive1a(a uint32, n uint32) uint32 {
	var b uint32
	if n == 0 {
		return 1
	} else {
		b = Mult (PowerRecursive1a(a, n/2), PowerRecursive1a(a, n/2))
		if n%2 == 0 {
			return b
		} else {
			return Mult(b, a)
		}
	}
}

func PowerRecursive2(a uint32, n uint32) uint32 {
	var b uint32
	if n == 0 {
		return 1
	} else {
		b = Mult(a, a) // b = a^2
		b = PowerRecursive2(b, n/2)
		if n%2 == 0 {
			return b
		} else {
			return Mult(b, a)
		}
	}
}
```

```
68  func PowerCyclic2(a uint32, n uint32) uint32 {
69        var res uint32 = 1
70        for n > 0 {
71              if n%2 == 1 {
72                    res = Mult(res, a)
73              }
74              n /= 2
75              a = Mult(a, a)
76        }
77        return res
78  }
79
80  func BenchmarkPowerCyclic0_6(b *testing.B) {
81      for i := 0; i < b.N; i++ {
82          PowerCyclic0(2020, 1000000)
83      }
84  }
85
86  func BenchmarkPowerRecursive0_6(b *testing.B) {
87      for i := 0; i < b.N; i++ {
88          PowerRecursive0(2020, 1000000)
89      }
90  }
91
92  func BenchmarkPowerRecursive1_6(b *testing.B) {
93      for i := 0; i < b.N; i++ {
94          PowerRecursive1(2020, 1000000)
95      }
96  }
97
98  func BenchmarkPowerRecursive1_9(b *testing.B) {
99      for i := 0; i < b.N; i++ {
100         PowerRecursive1(2020, 1000000000)
101     }
102 }
103
104 func BenchmarkPowerRecursive1a_6(b *testing.B) {
105     for i := 0; i < b.N; i++ {
106         PowerRecursive1a(2020, 1000000)
107     }
108 }
109
110 func BenchmarkPowerRecursive2_6(b *testing.B) {
111     for i := 0; i < b.N; i++ {
112         PowerRecursive2(2020, 1000000)
113     }
114 }
115
116 func BenchmarkPowerRecursive2_9(b *testing.B) {
117     for i := 0; i < b.N; i++ {
118         PowerRecursive2(2020, 1000000000)
119     }
120 }
121
122 func BenchmarkPowerCyclic2_6(b *testing.B) {
123     for i := 0; i < b.N; i++ {
124         PowerCyclic2(2020, 1000000)
125     }
126 }
127
128 func BenchmarkPowerCyclic2_9(b *testing.B) {
129     for i := 0; i < b.N; i++ {
130         PowerCyclic2(2020, 1000000000)
131     }
132 }
```

benchmark_Power_test.go

```
1   package main
2
3   import "fmt"
4
5   type arr [1000000]int64
6
7   func UnlimitedRecursion(a arr) {
8       fmt.Println(a[0])
9       a[0]++
10      UnlimitedRecursion(a)
11  }
12
13  func main() {
14        var c arr
15        UnlimitedRecursion(c)
16  }
```

stack_overflow.go

```
1   package main
2
3   import "fmt"
4
5   func P(n byte) {
6       if n>0 {
7           P(n / 2)
8           fmt.Printf("%d.", n)
9           P(n / 3)
10      }
11  }
12
13  func main() {
14      P(20)
15  }
```

abstractSample.go